CAPÍTULO 4

ESTRUTURA DO SISTEMA OPERACIONAL

Prezado aluno,

Neste capítulo você entenderá como o sistema operacional é estruturado e, que uma das principais preocupações em ambientes multitarefas é prover segurança. Aprenderá o que são chamadas de sistema e também o que são modos de acesso do processador. Em seguida, verá como estes dois conceitos são utilizados de forma conjunta para fornecer proteção ao processador e aos periféricos do sistema. Verá também, como a proteção de memória é realizada através de um hardware especial. E por último, as diferentes arquiteturas de sistemas operacionais bem como suas vantagens e desvantagens.

É importante entender como é realizada a proteção do sistema e como o hardware trabalha de forma conjunta com o sistema operacional para auxiliá-lo nesta tarefa. Além disso, entenda as diferentes arquiteturas de sistemas operacionais e descubra qual delas pode ser tendência para os próximos sistemas operacionais.

Bom estudo!

Fala Professor

4.1. Introdução

O sistema operacional é formado por um conjunto de rotinas que oferecem serviços aos usuários, às suas aplicações, e também ao próprio sistema. Esse conjunto de rotinas é denominado ***núcleo do sistema*** ou ***kernel***. É importante não confundir o núcleo do sistema operacional com aplicações e utilitários (interface gráfica e interpretador de comandos) que acompanham o sistema operacional.

Figura 4-17: Posicionamento do núcleo do sistema operacional em um sistema computacional
Fonte: [1] – Machado e Maia, 2004. Adaptação.

O sistema operacional é diferente de uma aplicação sequencial com início, meio e fim. Os procedimentos do sistema operacional são executados concorrentemente sem uma ordem pré-definida, com base em eventos assíncronos. Muitos desses eventos estão relacionados ao hardware e a tarefas internas do próprio sistema operacional.

As principais funções do núcleo do sistema operacional estão listadas abaixo:

- Tratamento de interrupções e exceções;
- Criação e eliminação de processos e threads;
- Sincronização e comunicação entre processos e threads;
- Gerência de memória;
- Gerência do sistema de arquivos;
- Gerência dos dispositivos de E/S;
- Suporte a redes locais e distribuídas;
- Contabilização do uso do sistema;
- Auditoria e segurança do sistema.

Obviamente, a estrutura do sistema operacional (a maneira como o código do sistema é organizado e o inter-relacionamento entre seus diversos componentes) pode variar conforme a concepção do projeto.

Existem diversas maneiras de estruturar o sistema operacional e estas serão vistas mais à frente. A seguir, serão apresentados os conceitos de chamadas de sistema e os modos de acesso.

4.2 Chamadas de sistema *(System Calls)*

Uma das preocupações importantes nos projetos de sistemas operacionais multiprogramáveis é prover compartilhamento de recursos de forma organizada e protegida. A eficiência proporcionada por um ambiente multiprogramável resulta em maior complexidade do Sistema Operacional (SO), já que diversos problemas de proteção surgem em decorrência desse tipo de implementação.

Como vários programas utilizam a memória simultaneamente, o SO deve evitar que um programa acesse a área do outro. De forma semelhante, o SO deve garantir a integridade e confidencialidade dos arquivos armazenados no disco. Além disso, o SO deve permitir que vários programas compartilhem o processador e impedir que um programa monopolize seu uso. Para solucionar tais problemas, o sistema operacional deve impedir que os programas de usuário acessem diretamente os recursos de hardware. Isto é realizado através das chamadas de sistema e dos modos de acesso do processador (que serão vistos mais à frente).

Conceitos

Chamadas de sistema (System Calls) são solicitações de recursos de hardware realizadas por utilitários e aplicativos ao Sistema Operacional [1].

Através dos modos de acesso, os programas de usuário (utilitários ou aplicações) são impedidos de acessarem diretamente os recursos de hardware. Assim, sempre que um programa de usuário desejar obter algum recurso de hardware, deverá realizar uma chamada de sistema. O SO recebe a chamada de sistema e decide se o recurso deve ser fornecido ou não ao programa.

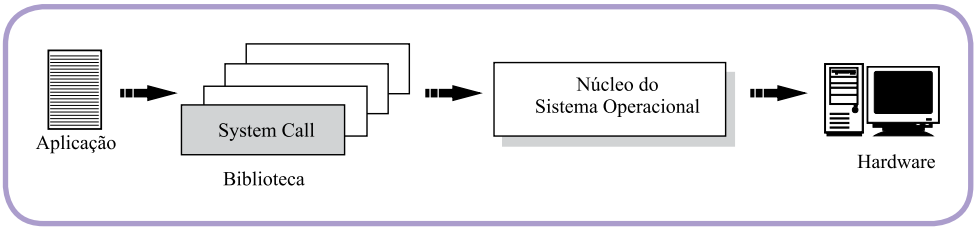

Figura 4-18: Chamada de sistema (system call)
Fonte: [1] – Machado e Maia, 2004. Adaptação.

As chamadas de sistema são implementadas pelo kernel do sistema operacional. Elas escondem os detalhes de implementação de hardware, checam a ocorrência de erros e retornam os dados ao usuário. Assim, os programadores e usuários não precisam saber, por exemplo, em que trilha e setor se encontra um determinado arquivo, pois isso será feito de forma transparente pelas chamadas de sistema. As chamadas de sistema podem ser agrupadas segundo suas funções: (i) gerência de processos e threads, (ii) gerência de memória, (iii) gerência do sistema de arquivos e (iv) gerência de dispositivos.

Para cada serviço oferecido há uma chamada de sistema associada e cada sistema operacional tem seu próprio conjunto de chamadas, com nomes, parâmetros e formas de ativação específicas. Isto explica porque uma aplicação desenvolvida utilizando serviços de um determinado sistema operacional não pode ser portada diretamente para outro sistema. Através dos parâmetros fornecidos na chamada de sistema, a solicitação é processada e uma resposta é retornada a aplicação juntamente com o estado de conclusão indicando se houve algum erro.

Os institutos ISO e IEEE propuseram uma interface de chamadas de sistema padronizadas denominada de padrão POSIX (*Portable Operating System Interface for Unix*). A ideia era que qualquer aplicação, seguindo este conjunto de chamadas padronizadas, pudesse ser executada em qualquer sistema que oferecesse suporte ao padrão. Inicialmente voltado para a unificação das diversas versões do Unix, o POSIX foi posteriormente incorporado pela maioria dos sistemas operacionais modernos. O termo *System Call* é típico de sistemas Unix, porém, em outros sistemas, o mesmo conceito é apresentado com diferentes nomes, como: *System Services* no OpenVMS e *Aplication Program Interface* (API) no Windows da Microsoft [1].

4.3. Modos de acesso

Para garantir a segurança do sistema, certas instruções – que podem comprometer a segurança e integridade do sistema – não podem ser executadas por programas de usuários (aplicações e utilitários). Em outras palavras, tais instruções devem ser executadas apenas pelo sistema operacional. Estas instruções são conhecidas como **instruções privilegiadas**. Por sua vez, as instruções que não oferecem risco ao sistema são denominadas **instruções não-privilegiadas** [1].

Para que somente o sistema operacional possa executar as instruções privilegiadas, o processador deve possuir um mecanismo de proteção conhecido como *modos de acesso*. Basicamente, na maioria dos processadores existem dois modos de acesso: **modo usuário** e **modo *kernel*** (ou **supervisor**) [1].

Quando o processador trabalha no modo usuário, o programa pode executar apenas instruções não-privilegiadas, não tendo acesso às instruções que podem comprometer o sistema. Já no modo *kernel* ou supervisor, o programa tem acesso a todas as instruções do processador, podendo executar tanto instruções privilegiadas quanto as não-privilegiadas. Assim, os programas de usuário executam no modo usuário, enquanto que o sistema operacional, executa no modo kernel.

O modo de acesso corrente (do programa que está executando no momento), é dado por um ou mais bits no registrador de status do processador. Através desse registrador, cada instrução a ser executada é verificada – por um hardware específico – se pode ou não ser executada. Caso o programa esteja no modo usuário e tente executar diretamente uma instrução privilegiada, será gerado um erro de proteção. O processador sinalizará este erro através de uma exceção, o sistema operacional será chamado e o programa será finalizado.

Fala Professor

Você já viu aquela mensagem no Windows: "Este programa executou uma operação ilegal e será fechado"? Ela indica que a aplicação causou um erro de proteção e o sistema operacional tratou corretamente a exceção, encerrando a aplicação. Se essa mensagem ocorrer, lembre-se: o erro foi da aplicação e não do sistema operacional.

Por outro lado, se a tela azul aparecer indica que o sistema operacional se tornou instável – algo que nunca deveria acontecer com nenhum sistema operacional, fazer o quê?! Neste caso, é recomendável reiniciar a máquina para evitar problemas.

Você deve estar se perguntando como ocorrem as transições entre os modos de acesso, ou seja, do modo usuário para modo *kernel* e vice-versa. No momento da carga do sistema (*boot*), o sistema operacional inicia em modo kernel. Após estar carregado em memória, o sistema operacional permite que os programas de usuários sejam carregados apenas em modo usuário.

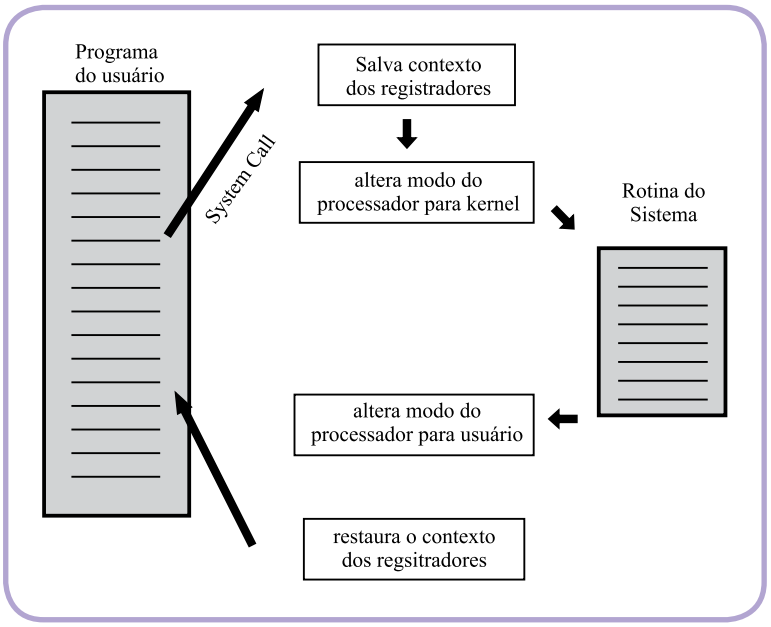

Figura 4-19: Mudança dos modos de acesso durante uma interrupção
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Se um programa estiver executando em modo usuário e ocorrer qualquer tipo de interrupção (interrupção de hardware, exceção ou interrupção de software), o modo de acesso é alterado para modo *kernel*. Com isso, a rotina de tratamento é executada em modo *kernel*. Ao final de toda rotina de tratamento, há uma instrução específica que, antes de retornar para o programa do usuário, altera o modo de acesso para modo usuário.

Obviamente, as etapas de salvar o contexto de execução, identificar a origem do evento, obter o endereço da rotina de tratamento, e, restaurar o contexto de execução são sempre realizadas durante uma interrupção. O destaque aqui é que antes de executar a rotina de tratamento deve haver uma mudança para o modo kernel e, após sua execução, deverá haver a volta para o modo usuário.

4.4. Proteção do sistema

Para realizar a proteção de um sistema de computação é necessário proteger os periféricos, o processador e a memória de uma má utilização por parte dos programas de usuários. A seguir, veremos como são implementadas tais proteções.

4.4.1. Proteção dos periféricos

Para proteger os periféricos, os programas de usuário devem ser impedidos de acessar diretamente os dispositivos de E/S. Somente o sistema operacional deve acessar diretamente as instruções de E/S. Para implementar isso, todas as instruções de E/S devem ser privilegiadas. Assim, os programas de usuário não poderão executar diretamente tais instruções. A única maneira que os programas de usuário poderão realizar E/S será através de chamadas de sistema.

4.4.2. Proteção do processador

Em sistemas multiprogramáveis, os programas compartilham a utilização do processador. Isto é feito por permitir que cada programa utilize o processador por um determinado tempo (este tempo é chamado de *time slice*). A configuração desse intervalo de tempo é feita no registrador do temporizador (*timer* – hardware que gera interrupções em intervalos de tempo pré-configurados). Se um programa de usuário tiver privilégio para executar instruções que alterem o *time slice*, ele poderá monopolizar o processador. Assim, para impedir que isso ocorra, todas as instruções relacionadas com a configuração do temporizador devem ser privilegiadas.

4.4.3. Proteção da memória

Para que a memória seja utilizada por diversos programas de forma protegida, um programa não deve ter acesso à área de memória de outros, e sim, somente à sua própria área de memória. Esta proteção é realizada através de um hardware específico. Para saber qual a área alocada por um programa, esse hardware possui dois registradores (base e limite) que determinam as posições inicial e final da memória alocada.

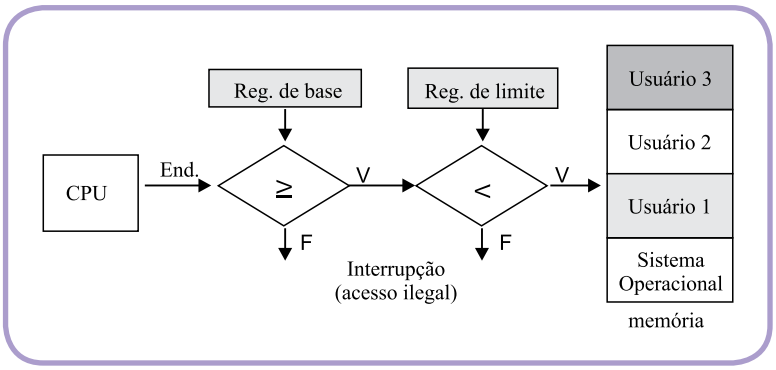

Figura 4-20: Hardware de proteção da memória
Fonte: [4] – Oliveira, Carissimi e Toscani, 2004. Adaptação

Quando um programa tentar acessar algum endereço de memória, o hardware verificará se o endereço está entre as posições inicial e final, e permitirá o acesso somente em caso afirmativo. Caso contrário, será gerada uma exceção de proteção (indicando acesso ilegal à memória), o sistema operacional será chamado e finalizará o programa.

Atividades

Atividades

1. O que é o núcleo do sistema e quais são suas principais funções?
2. O que é uma *system call* e qual sua importância para a segurança do sistema? Como as *system calls* são utilizadas por um programa?
3. O que são instruções privilegiadas e não privilegiadas? Qual a relação dessas instruções com os modos de acesso?
4. Quais das instruções a seguir devem ser executas apenas em modo kernel?

a) Desabilitar todas as interrupções;

b) Consultar a data e hora do sistema;

c) Alterar a data e hora do sistema;

d) Alterar informações residentes no núcleo do sistema;

e) Somar duas variáveis declaradas dentro do programa;

f) Realizar um desvio para uma instrução dentro do próprio programa e acessar diretamente posições no disco.

5. Explique como funciona a mudança de modos de acesso e dê um exemplo de como um programa faz uso desse mecanismo.
6. Como o kernel do sistema operacional pode ser protegido pelo mecanismo de modos de acesso?

4.5 Arquitetura

4.5.1. Arquitetura monolítica

Na arquitetura monolítica, os módulos do sistema operacional são compilados e linkados em um único e grande programa executável. Todos os componentes do sistema operacional (agendamento de processos, gerenciamento de memória, operações de entrada e saída, acesso ao sistema de arquivos) estão contidos no mesmo espaço de endereçamento do núcleo. Assim, tais componentes podem comunicar-se diretamente, e por isso, esta arquitetura apresenta um excelente desempenho.

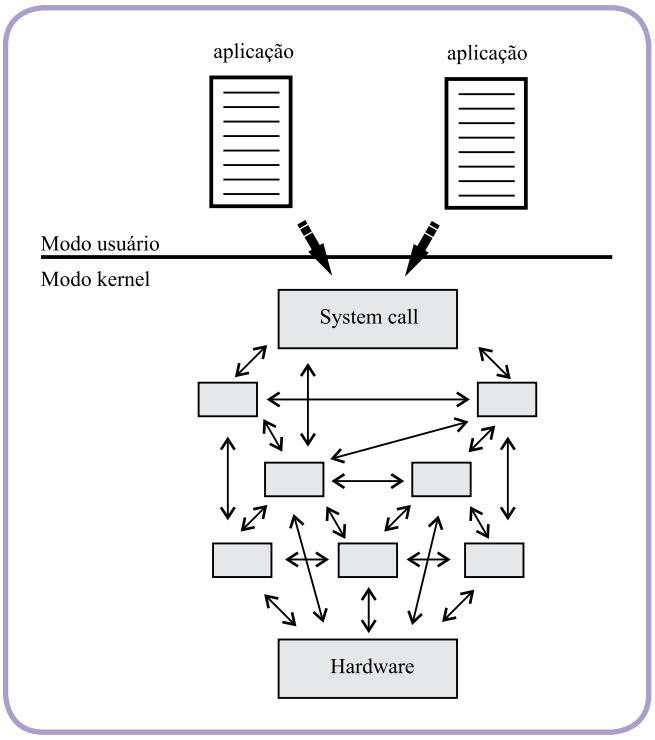

Figura 4-21: Arquitetura monolítica
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Porém, o fato de todos os componentes estarem juntos em um único programa executável dificulta bastante a manutenção do código e a correção de erros. Além disso, um erro em algum módulo pode comprometer todos os demais. Os primeiros sistemas operacionais utilizavam esta arquitetura, que devido a sua simplicidade e bom desempenho, foi utilizada no projeto dos primeiros sistemas Unix e do MS-DOS.

À medida que os sistemas operacionais tornaram-se mais complexos, projetos puramente monolíticos tornaram-se inviáveis e precisaram ser reestruturados. Uma das variações que surgiu com a arquitetura monolítica foi permitir que módulos fossem adicionados dinamicamente ao kernel.

4.5.2. Arquitetura de camadas

Com o aumento da complexidade e do tamanho do código dos sistemas operacionais, técnicas de programação estruturada e modular foram incorporadas ao seu projeto. Na *arquitetura em camadas*, o sistema é dividido em níveis sobrepostos. Cada camada oferece um conjunto de funções que podem ser utilizadas apenas pelas camadas superiores, onde a camadas mais baixa presta serviços à camada de cima.

Neste tipo de implementação, as camadas mais internas são mais privilegiadas que as camadas mais externas. Além disso, as camadas são modulares, ou seja, uma camada pode ser alterada sem exigir alteração nas demais. A vantagem da estruturação em camadas é isolar as funções do sistema operacional, facilitando sua manutenção e depuração, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas.

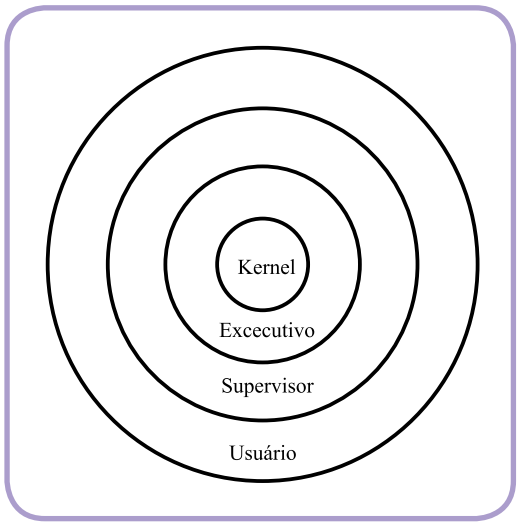

Figura 4-22: Exemplo de arquitetura em quatro camadas (OpenVMS)
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Uma desvantagem é a queda no desempenho, pois cada nova camada implica em uma mudança no modo de acesso. Atualmente, a maioria dos sistemas comerciais utiliza o modelo de duas camadas, onde existem os modos de acesso kernel e usuário. A maioria das versões do Unix e do Windows 2000 é baseada neste modelo.

4.5.3. Arquitetura de máquina virtual

Um sistema computacional é formado por níveis, onde a camada de nível mais baixo é o hardware. Acima desta camada está o sistema operacional que oferece suporte às aplicações. O modelo de **Máquina Virtual** (VM - *Virtual Machine*), cria um nível intermediário entre o hardware e o sistema operacional, denominado gerência de máquinas virtuais. Nes-

te nível são criadas máquinas virtuais independentes, onde cada uma oferece recursos de hardware virtuais, incluindo os modos de acesso, interrupções, dispositivos de E/S, etc.

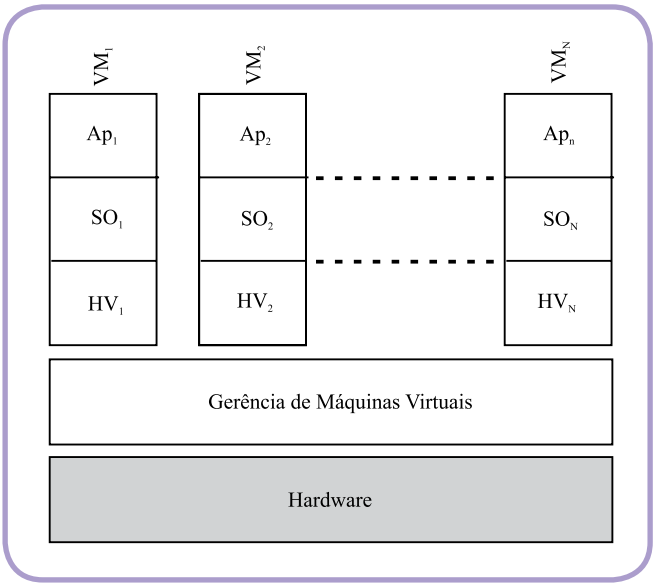

Figura 4-23: Máquina virtual
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Visto que cada máquina virtual é independente das demais, é possível que cada VM tenha seu próprio sistema operacional e que seus usuários executem suas aplicações como se todo o computador estivesse dedicado a cada um deles. Além de permitir que vários sistemas operacionais sejam executados no mesmo computador, este modelo cria o isolamento total entre cada VM, oferecendo grande segurança para cada máquina virtual. Por exemplo, se uma VM executar uma aplicação que comprometa o funcionamento do seu sistema operacional, as demais máquinas não sofrerão qualquer problema. A desvantagem desta arquitetura é a sua grande complexidade, devido à necessidade de se compartilhar e gerenciar os recursos de hardware entre as diversas VM's.

Outro exemplo de utilização desta arquitetura ocorre na máquina virtual Java, desenvolvida pela Sun Microsystems. Para se executar um programa em Java é necessário que uma máquina virtual Java (JVM – *Java Virtual Machine*) esteja instalada no sistema. Qualquer sistema operacional pode suportar uma aplicação Java, desde que exista uma JVM desenvolvida para ele.

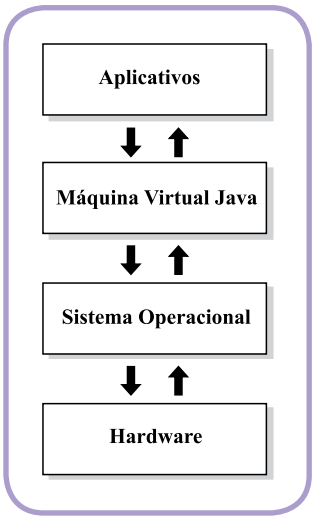

Figura 4-24: Máquina virtual Java
Fonte: [1] – Machado e Maia, 2004. Adaptação.

A aplicação Java deve ser compilada para a linguagem de binária da máquina virtual Java (JVM). Desta forma, a aplicação não precisa ser recompilada para cada sistema computacional diferente, tornando-se independente do hardware e sistema operacional utilizados. A grande vantagem deste modelo é a portabilidade. A desvantagem é o seu menor desempenho se comparada a uma aplicação compilada e executada diretamente em uma arquitetura específica.

4.5.4. Arquitetura Microkernel

A arquitetura Microkernel procura tornar o núcleo do sistema operacional menor e mais simples possível. Para isso, os serviços do sistema são disponibilizados através de processos, onde cada um é responsável por oferecer um conjunto específico de funções, como gerência de arquivos, gerência de processos, gerência de memória e escalonamento.

Sempre que uma aplicação necessitar de algum serviço, será realizada uma solicitação ao processo responsável. Neste caso, a aplicação que solicita o serviço é chamada de cliente, enquanto o processo que responde à solicitação é chamado de servidor. Um cliente solicita um serviço enviando uma mensagem ao servidor. O servidor responde ao cliente através de outra mensagem. A principal função do núcleo é realizar a comunicação, ou seja, a troca de mensagens ente cliente e servidor.

A utilização deste modelo permite que os servidores executem em modo usuário, não tendo acesso direto a certos componentes do sistema. Apenas o núcleo do sistema, responsável pela comunicação entre clientes e servidores, executa no modo *kernel*. Devido a isso, se ocorrer um erro em um servidor, este poderá parar, mas o sistema não ficará inteiramente comprometido, aumentando assim a sua disponibilidade.

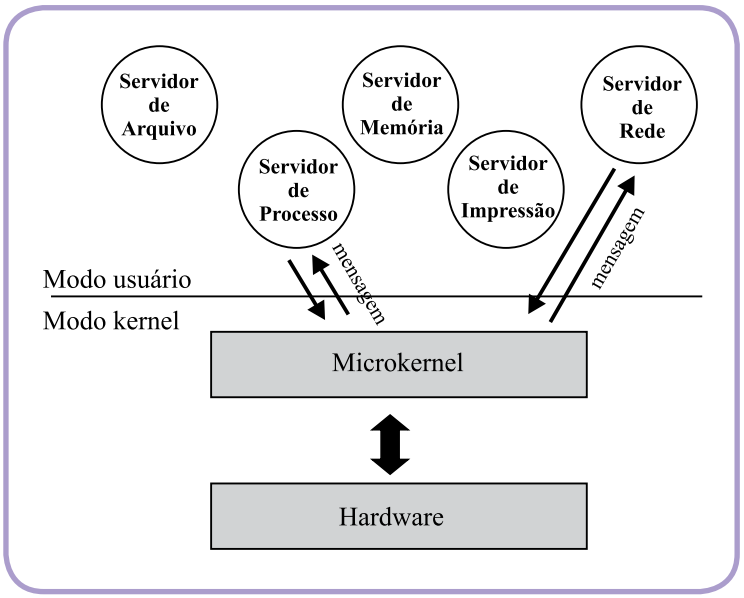

Figura 4-25: Arquitetura Microkernel
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Outra característica interessante é que, como os servidores se comunicam através da troca de mensagens, não importa se os clientes e servidores estão sendo processados em um sistema com um único processador, com múltiplos processadores (fortemente acoplado) ou ainda em ambiente de sistema distribuído (fracamente acoplado). Um sistema microkernel, implementado em ambientes distribuídos, permite que um cliente solicite um serviço e que a resposta seja processada remotamente. Esta característica permite acrescentar novos servidores à medida que o número de clientes aumenta, conferindo uma grande escalabilidade ao sistema operacional.

Além disso, a arquitetura microkernel permite isolar as funções do sistema operacional por diversos processos servidores, tornando o núcleo menor, mais fácil de depurar e, consequentemente, aumentando sua confiabilidade. Com isso, há um aumento da facilidade de manutenção, flexibilidade e portabilidade.

Apesar de todas as vantagens deste modelo, sua implementação, na prática, é muito difícil. Primeiro há o problema do desempenho, devido à necessidade de mudança de modo de acesso a cada comunicação entre clientes e servidores. Outro problema é que certas funções do sistema operacional exigem acesso direto ao hardware, como operações de E/S. Na realidade, o que é implementado mais usualmente é uma combinação do modelo de camadas com a arquitetura microkernel. O núcleo passa a incorporar outras funções críticas do sistema, como escalonamento, tratamento de interrupções e gerência de dispositivos.

Existem vários projetos baseados em sistemas microkernel, principalmente em instituições de ensino e centro de pesquisa, como o Exokernel

- do MIT (Massachusetts Institute of Technology); o L4 - da Universidade de Dresden; e o Amoeba - da Vrije Universiteit. A maioria das iniciativas, nesta área, está relacionada ao desenvolvimento de sistemas operacionais distribuídos.

Indicações

[1] MACHADO, F.B. e MAIA, L.P. ***Arquitetura de Sistemas Operacionais***. 4.ed. LTC, 2007.

[2] SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G. ***Fundamentos de Sistemas Operacionais***. 6.ed. LTC, 2004.

[3] TANENBAUM, A.S. ***Sistemas Operacionais Modernos***. 2.ed. Pearson Brasil, 2007.

[4] OLIVEIRA, R.S., CARISSIMI, A.S., TOSCANI, S.S. ***Sistemas Operacionais***. 3.ed. Sagra-Luzzato. 2004.

Atividades

Atividades

1. Compare as arquiteturas monolíticas e de camadas. Quais as vantagens e desvantagens de cada arquitetura?
2. Quais as vantagens do modelo de máquina virtual?
3. Como funciona o modelo cliente-servidor na arquitetura microkernel? Quais as vantagens e desvantagens dessa arquitetura?
4. Pesquise na Internet três máquinas virtuais diferentes e compare-as.
5. Baixe e instale a máquina virtual VirtualBox no seu computador. Em seguida, consiga a última versão do Ubuntu GNU/Linux e instale nessa máquina.